Unity 3D

Présentation et Prise en main

guillaume.bouyer@ensiie.fr
www.ensiie.fr/~bouyer/

Unity 3D

Moteur de jeux vidéo

- Edition personnelle gratuite / Edition professionnelle payante
- Iteration rapide
- Multiplateforme
- Asset Store

Liens

- http://unity3d.com/unity/
- http://unity3d.com/learn
  - Manual
  - Scripting API
  - Tutorials
  - Resources

Moteur de jeu

Ensemble d'outils pour faire un ou plusieurs jeux

- Editeur de scène
- Moteur de rendu 3D
- Moteur physique
- Rendu audio
- Réseau
- Intelligence Artificielle
- Scripts : comportements, événements…
- Animation
- …

Exemples

Bad Piggiez (Rovio)

Temple Run 2 (Imangi Studios)

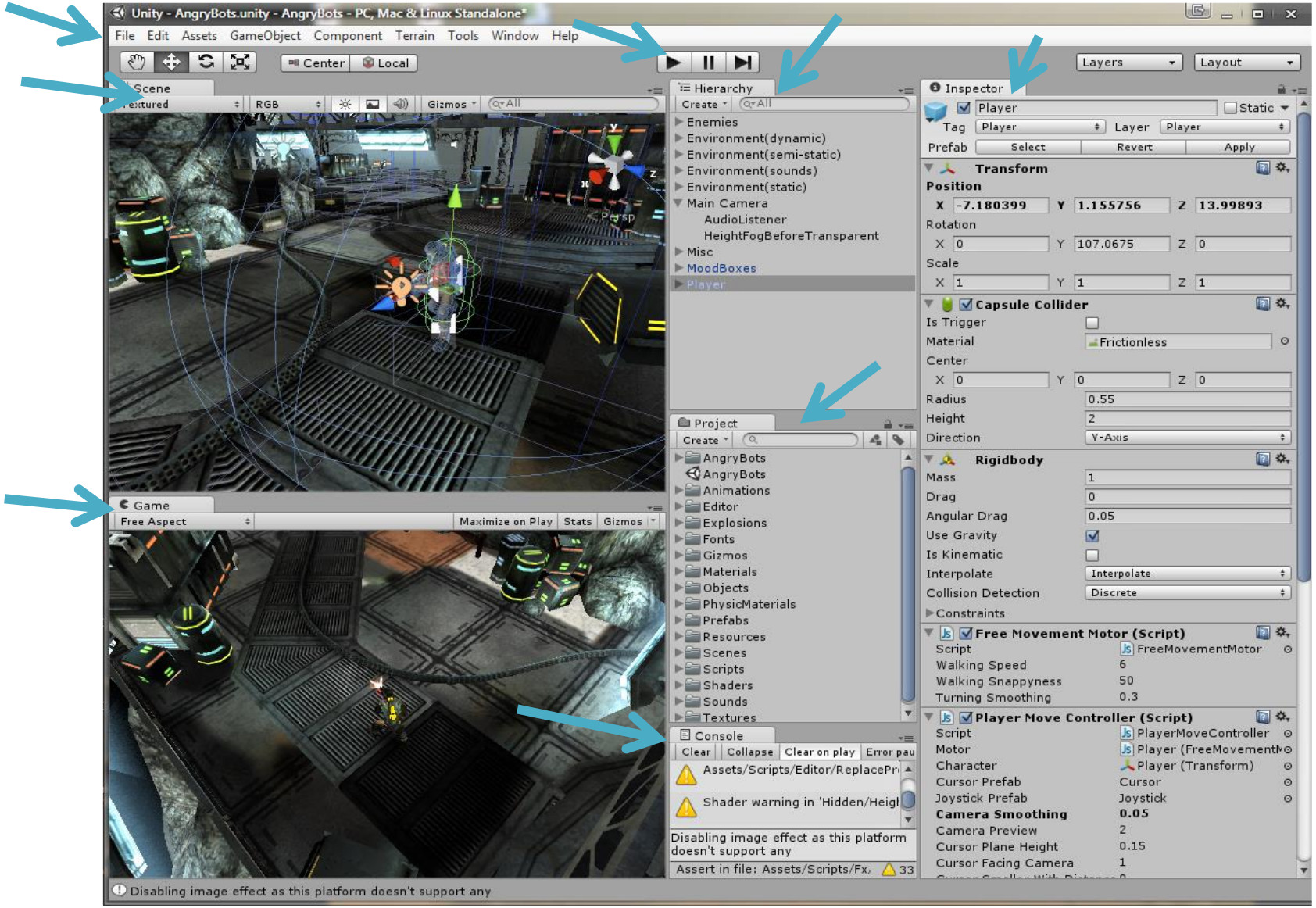
Endless Space, Endless Legends (Amplitude Studios)

…

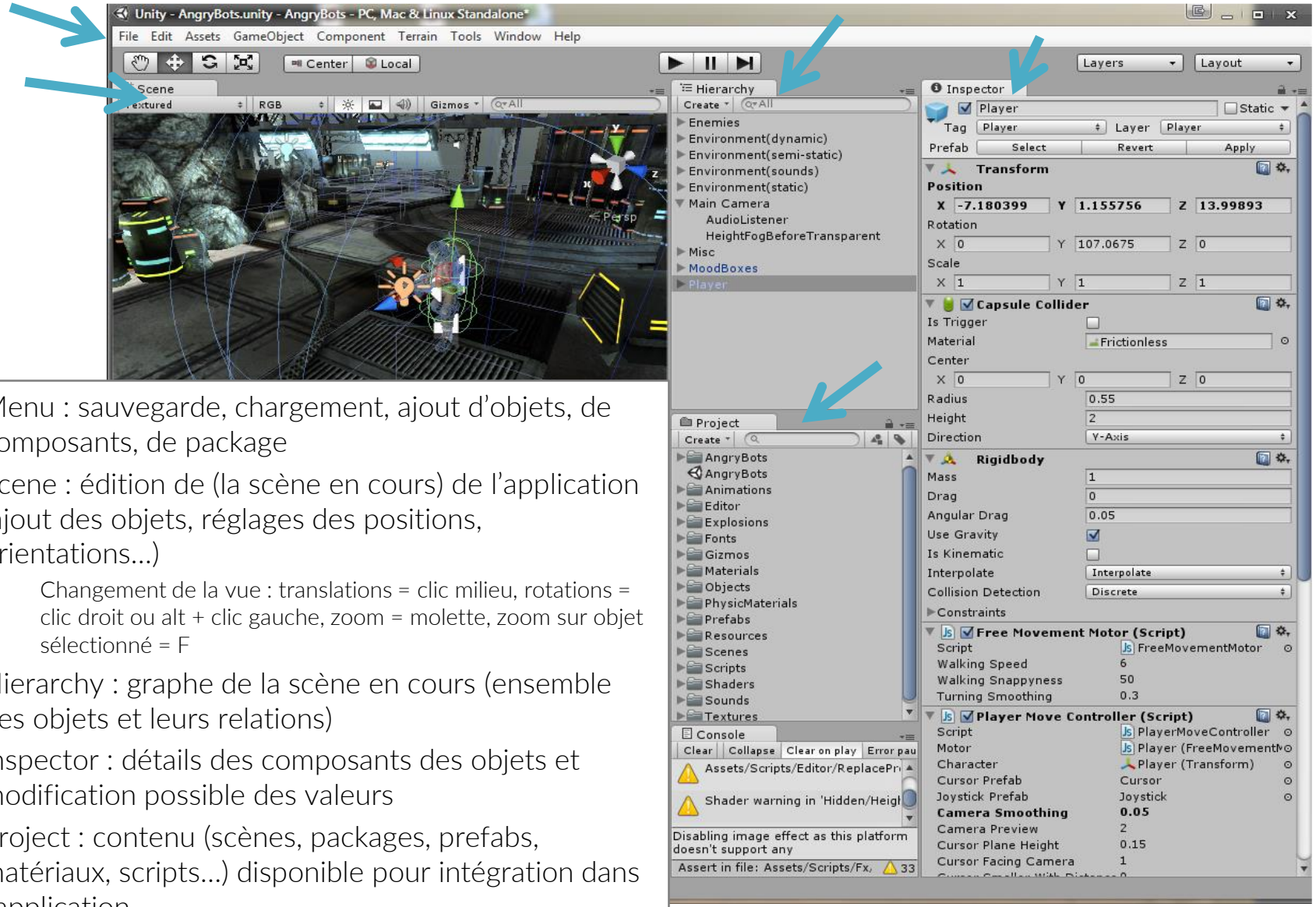
TP

Lancer Unity

Interface et organisation



Edition



Menu : sauvegarde, chargement, ajout d'objets, de composants, de package

Scene : édition de (la scène en cours) de l'application (ajout des objets, réglages des positions, orientations…)

> Changement de la vue : translations = clic milieu, rotations = clic droit ou alt + clic gauche, zoom = molette, zoom sur objet sélectionné = F

Hierarchy : graphe de la scène en cours (ensemble des objets et leurs relations)

Inspector : détails des composants des objets et modification possible des valeurs

Project : contenu (scènes, packages, prefabs, matériaux, scripts…) disponible pour intégration dans l'application

Exécution

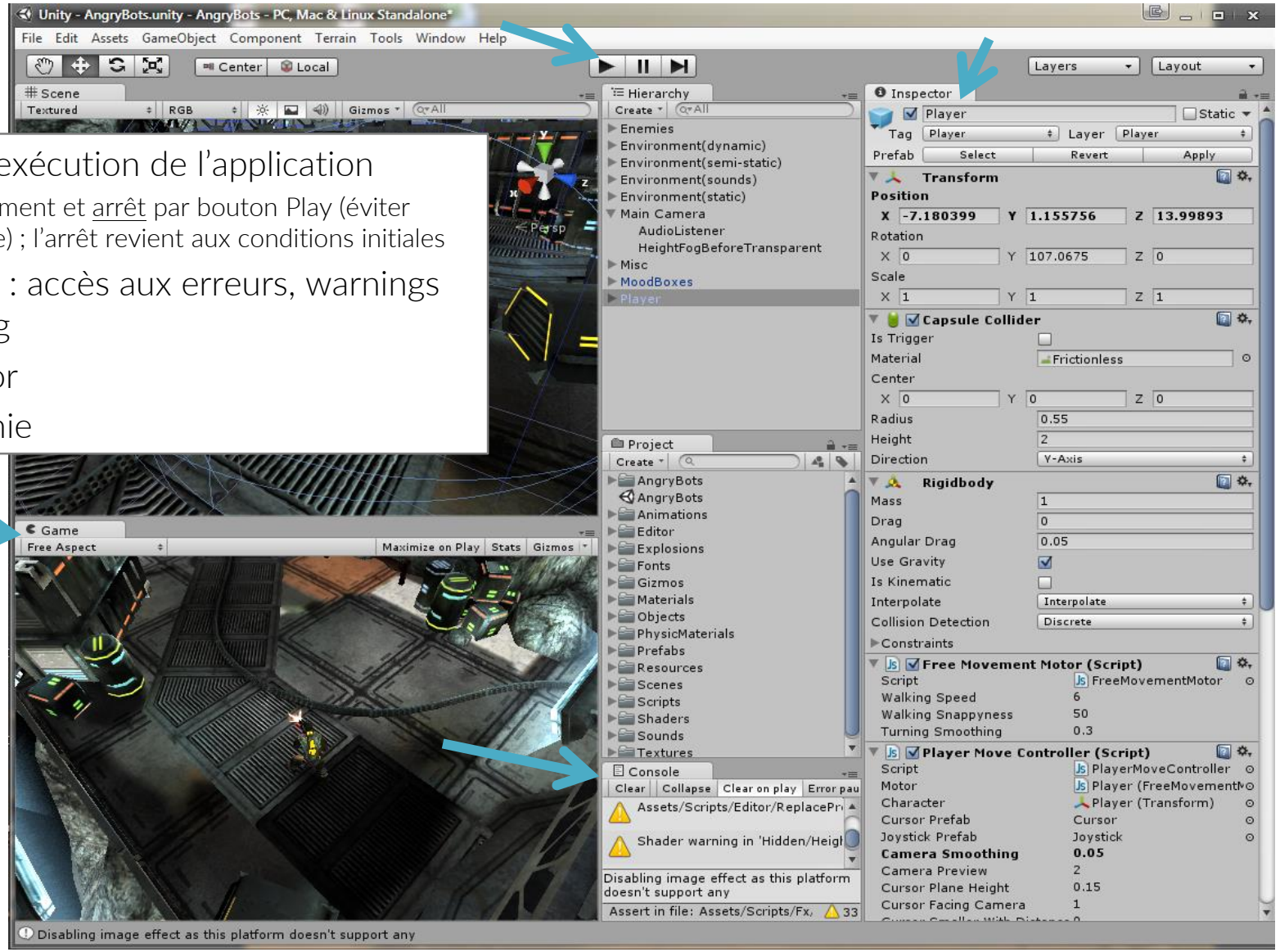
Game : exécution de l'application

lancement et arrêt par bouton Play (éviter Pause) ; l'arrêt revient aux conditions initiales

Console : accès aux erreurs, warnings et debug

Inspector

Hiérarchie



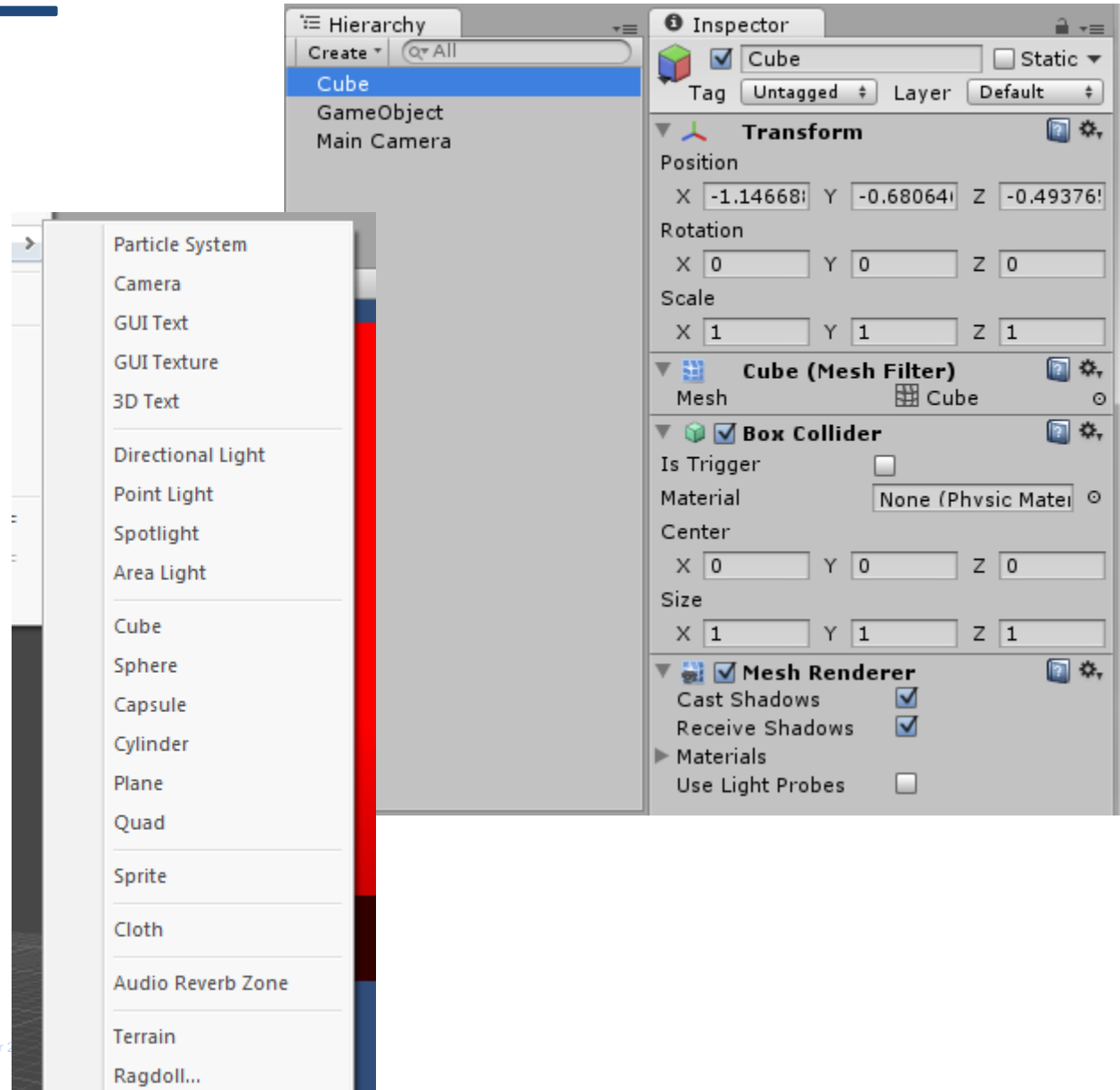
GameObjects et Components

Tous les objets utilisés dans l'application sont des GameObjects

Ils contiennent des propriétés appelées Components

Exemples :

- Un objet vide contient uniquement un composant *Transform* permettant de gérer ses propriétés spatiales (sa position, son orientation et son échelle)
- Un solide simple possèdera en plus un *Mesh Filter* (géométrie), un *Mesh Renderer* (rendu visuel) et un *Collider* (collisions)



GameObjects et Components

Les composants sont le cœur de la « programmation » de l'application

Ils donnent des propriétés et des comportements aux objets de la scène.

- On peut en ajouter à chaque objet via le menu Component : *Effect*, *Physics*…
- Ils peuvent être édités dans l'Inspector, les possibilités variant en fonction des champs : booléens, listes, valeurs numériques, courbes…
- On peut les supprimer de l'objet par le menu clic droit (NB : perte de tous les réglages) ou les désactiver avec la case à cocher

On peut programmer ses propres composants grâce aux scripts

- Une fois associés à un objet, ils apparaissent sous la même forme que les autres composants (cf. Scripts)

TP

Construction du projet

Projet

Dossiers

Scène

Insérer un plan

OBJETS ET COMPOSANTS ESSENTIELS

Propriétés spatiales

Transform

Tout objet (même vide) contient au moins un composant Transform qui stocke sa position, son orientation et son échelle sous forme de vecteurs selon les axes XYZ (représentés visuellement par les flèches de couleur RGB).

Ces 3 propriétés peuvent être modifiées

à l'édition :

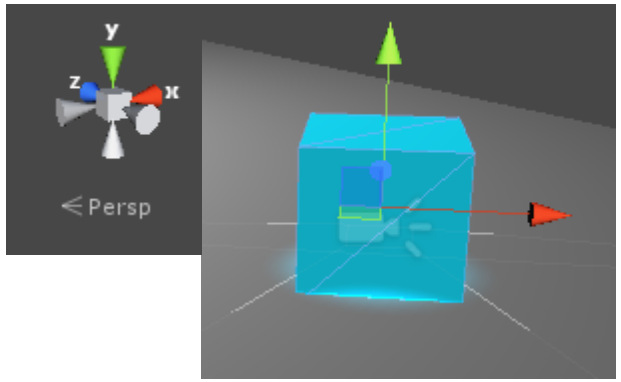
avec les outils souris

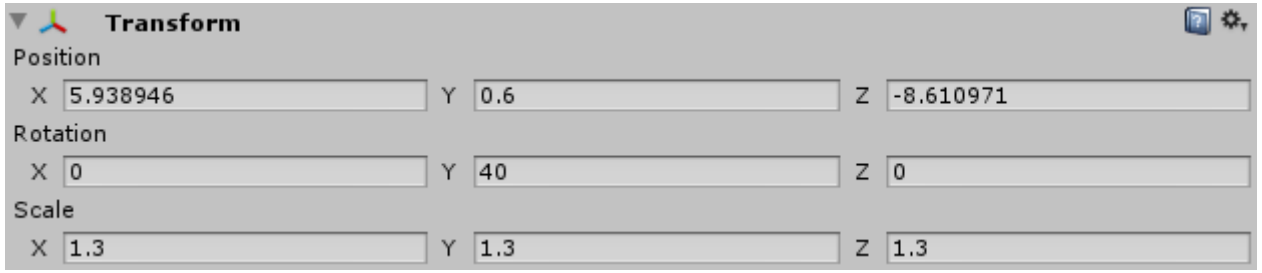
dans l'inspector.

à l'exécution (donc par script)

via le composant *transform*

et les variables/fonctions associées : *position*, *rotation*, *scale*, *translate()*, *rotate()*...





Propriétés spatiales

Graphe de scène

Organisation des différents objets de la scène dans une hiérarchie :

Chaque objet peut avoir un seul parent et plusieurs enfants.

Toute opération spatiale effectuée sur un objet est répercutée sur ses enfants.

L'objet en haut de la hiérarchie est la racine.

Coordonnées locales vs. globales

Locales = définies par rapport à son parent

Globales = par rapport à l'origine de la scène

Le *transform* d'un objet dans l'Inspector est local

Par script on peut accéder aux 2

La hiérarchie se construit/modifie

à l'édition par glisser/déposer dans l'onglet de même nom

par script : `transform.parent = ...`

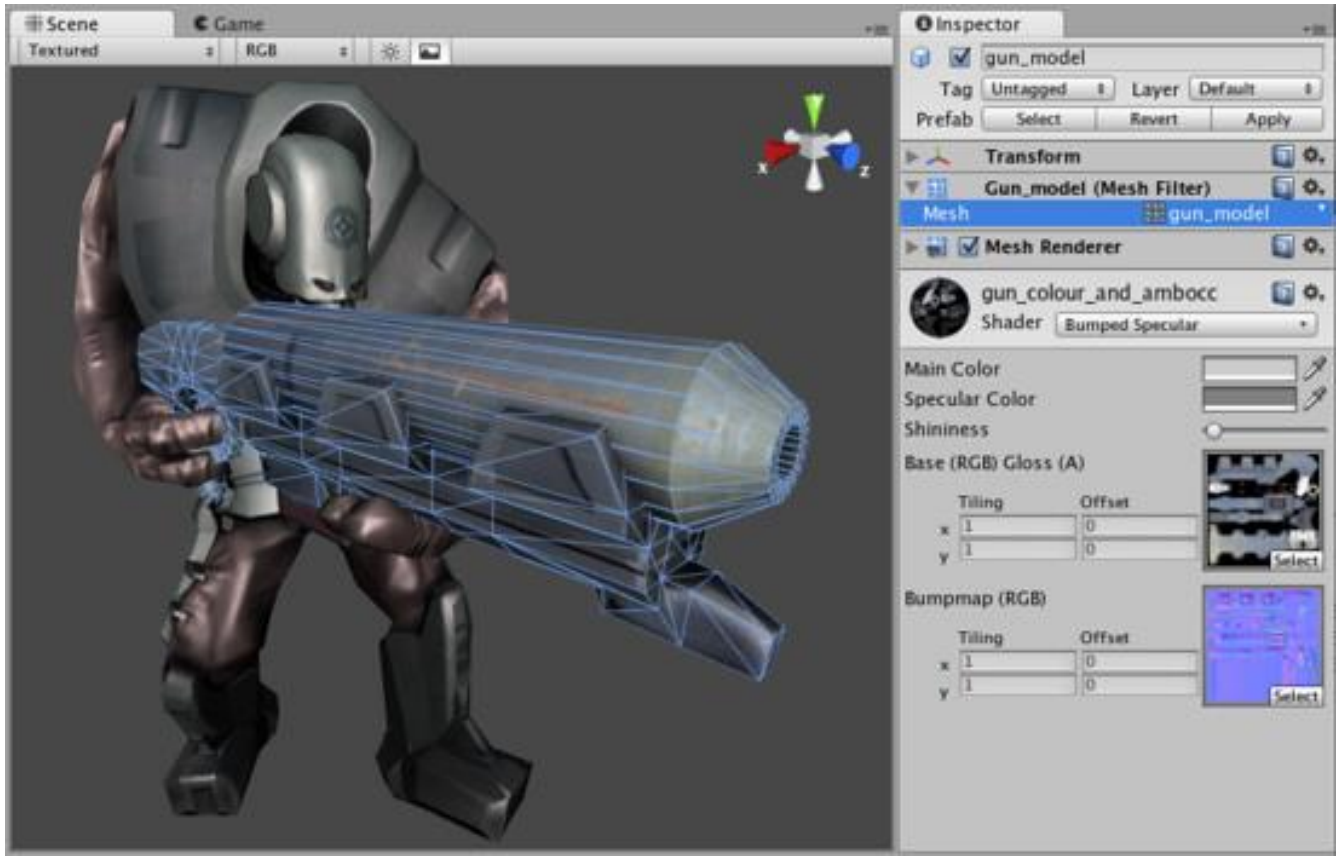
Propriétés spatiales

NB : Pour que votre programme n'ait pas de comportement imprévu, il est recommandé officiellement de toujours manipuler des modèles à une échelle uniforme (soit `scale = (1,1,1)` voire `(X,X,X)`).

-> Il faut donc créer ces modèles directement à la forme/taille voulue dans un modeleur et les importer sous Unity

Propriétés géométriques et visuelles

Mesh Filter (géométrie) + *Mesh Renderer* (rendu)



Propriétés géométriques et visuelles

Materials

Création : via le menu Assets ou par clic droit dans le projet -> Create > Material

Application à un objet par glisser-déposer

Textures

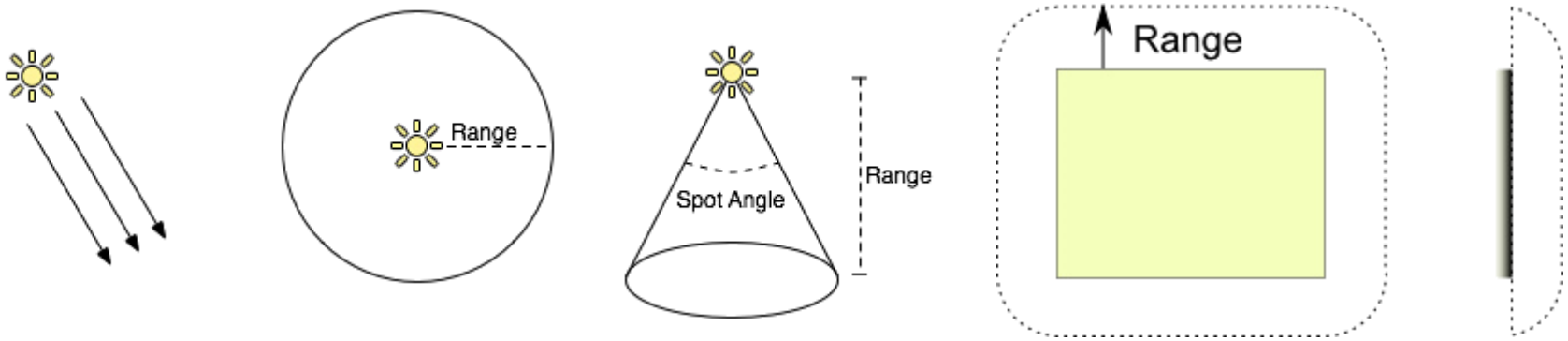
Importer l'image dans les assets

L'appliquer à un material

Shaders

Pré-installés ou à programmer (HLSL)

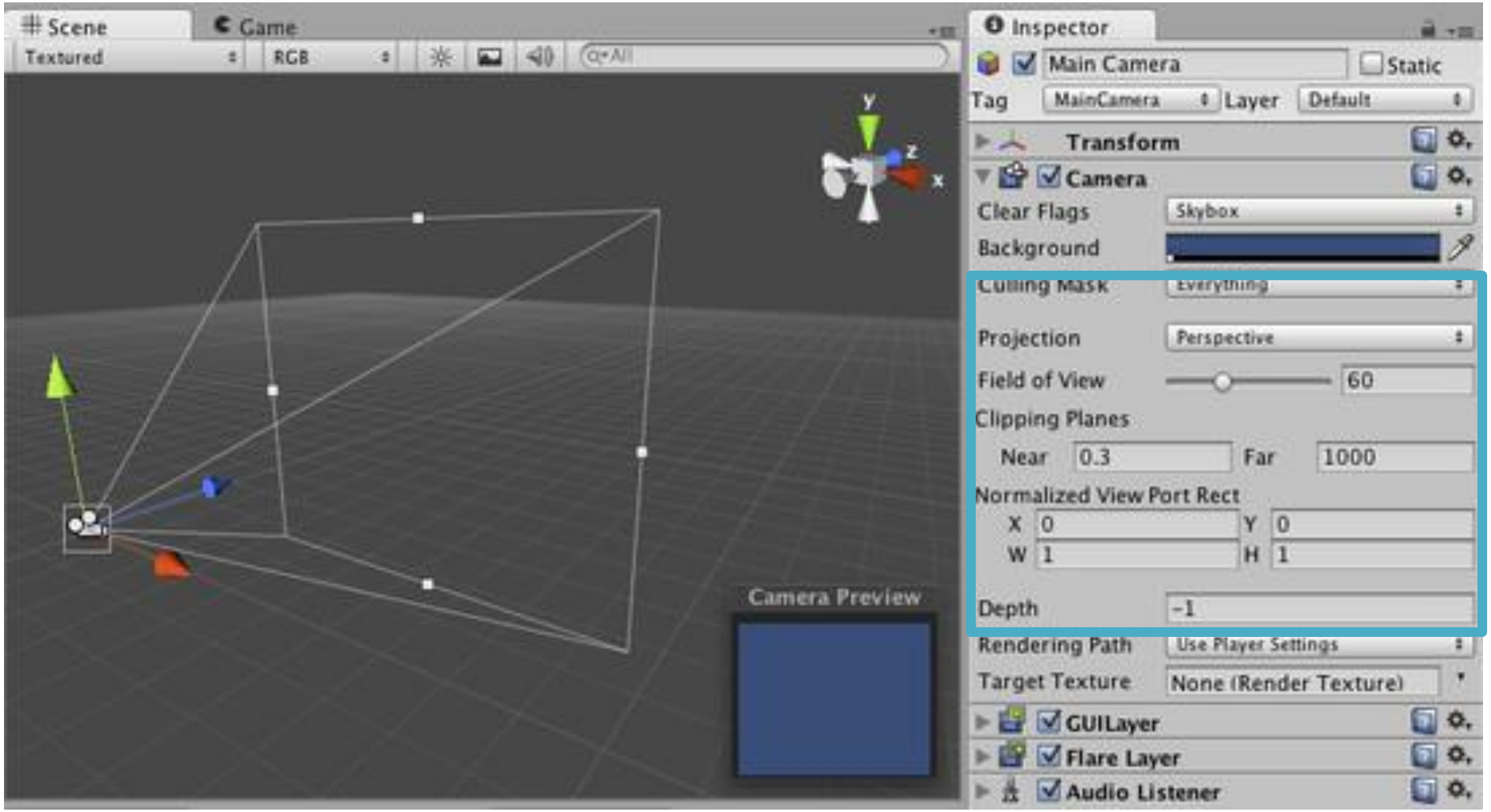
Lumières



Directional Point Spot Area

http://docs.unity3d.com/Manual/class-Light.html

Caméras



http://docs.unity3d.com/Documentation/Components/class-Camera.html
http://docs.unity3d.com/Manual/class-Camera.html

Prefabs

Les Prefabs sont des sortes de templates, des patrons d'objets qui peuvent être réutilisés pour créer d'autres objets similaires

On les obtient à partir d'un objet modèle que l'on crée puis que l'on fait glisser de la fenêtre hiérarchie vers la fenêtre projet

On peut ensuite par l'opération inverse créer à l'édition autant d'instances du Prefab que l'on veut

Il est aussi possible de créer des instances au runtime (cf. section)

Recopie/Modification de propriétés

Il y a « héritage » entre le prefab source et les instances. Toute modification de la source est répercutée sur les instances.
Il est par contre possible de
modifier une instance particulière (surcharge d'une propriété),
puis d'appliquer ces modifications aux autres (Apply),
ou de les annuler (Revert)

TP

Construction du projet

Light

Cube

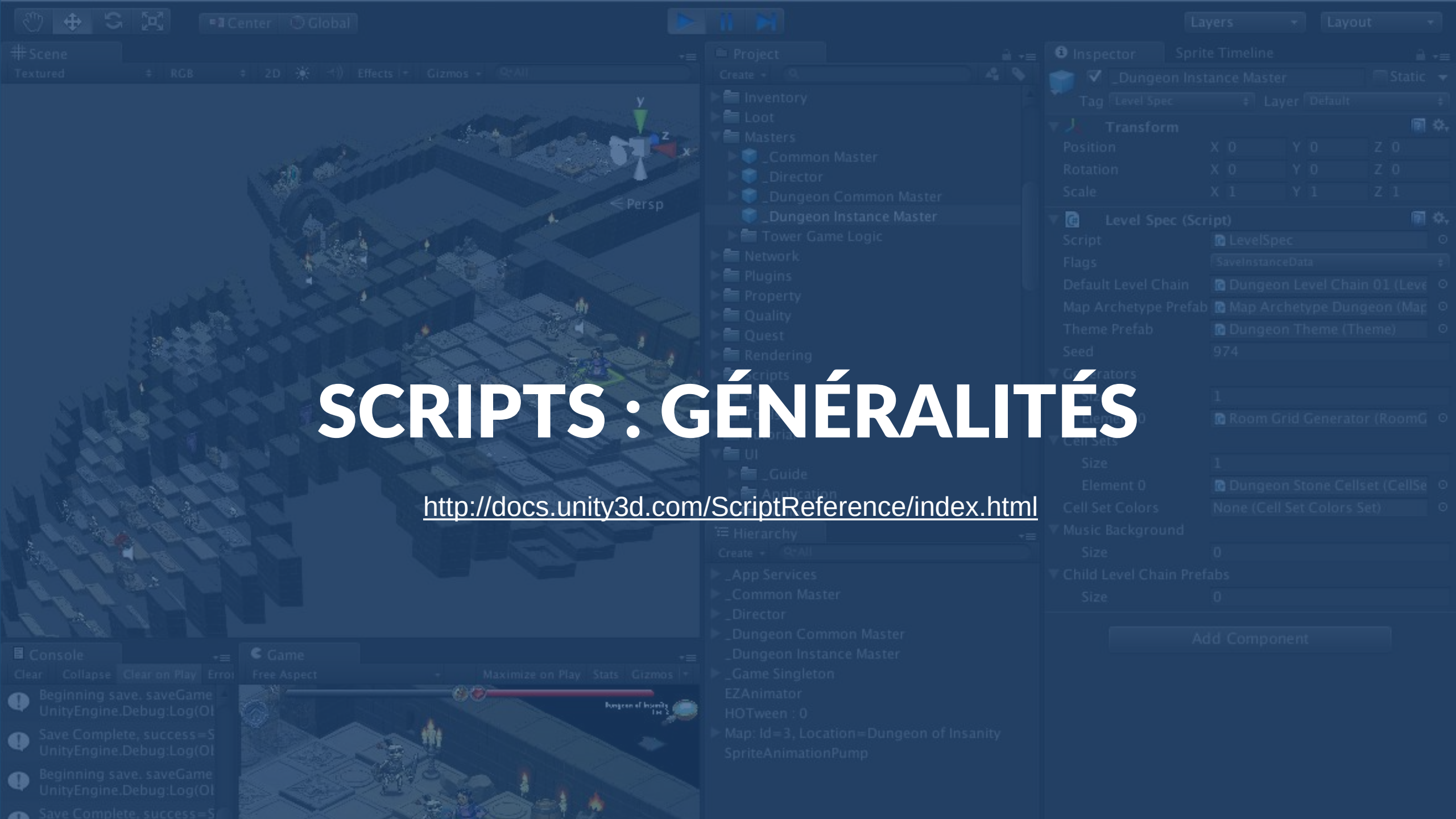
Material

Image

Prefab

SCRIPTS : GÉNÉRALITÉS

http://docs.unity3d.com/ScriptReference/index.html



Présentation

Langages supportés

- ~~Unityscript ≈ Javascript~~
- C#

Création d'un nouveau script

- Menu Assets
- Ou clic droit dans le projet --> Create --> <Langage> Script

Edition

- Avec Visual Studio ou MonoDevelop, installés avec Unity selon l'OS utilisé
- Ou avec tout autre éditeur (Visual studio…)

Script C# par défaut


```
using UnityEngine;
using System.Collections;

public class #SCRIPTNAME# : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
    }
}
```

Rq : template éditable Program Files\Unity\Editor\Data\Resources\ScriptTemplates

Utilisation

Par défaut, un script hérite de la classe MonoBehaviour

qui hérite de Behaviour (tout composant qui peut être activé ou désactivé)

qui hérite de Component (tout ce qui peut être attaché à un GameObject)

=> On relie un script MonoBehaviour à un objet comme tout autre composant

Les instructions dans le script vont par défaut s'appliquer à l'objet associé => choisir le propriétaire du script fait partie de la conception !

Rq : on peut écrire des classes qui ne sont pas des MonoBehaviour

Game Loop & Update

Boucle de mise à jour et de rendu du jeu en fonction des commandes du joueur

```
while (true) {
    ProcessInput();
    UpdateGame();
    RenderGame();
}
```

1 itération de la boucle => 1 image ("frame" => FramePerSecond)

Fonction de base d'un script : `Update()`

s'exécute à chaque frame de la Game Loop et permet de personnaliser la mise à jour de chaque composant de chaque objet

Exemple

```
void Update () {                  //A chaque frame
    transform.Rotate(0, 5, 0); //Appel à la méthode Rotate du composant transform de l'objet courant,
                               //qui effectue une rotation de cet objet de 5 degrés selon l'axe y
}
```

Rq : rotation de 300°/s si l'application tourne à 60 fps…

Autres fonctions

`Start()` s'exécute à l'activation du script : initialisations

- Nombreuses autres fonctions possibles, notamment celles qui s'exécutent suite à un événement particulier (cf Events)

```
OnMouseDown()
OnCollisionEnter()
OnApplicationQuitSent()
OnConnectedToServer()
...
```

http://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Modes d'accès aux variables

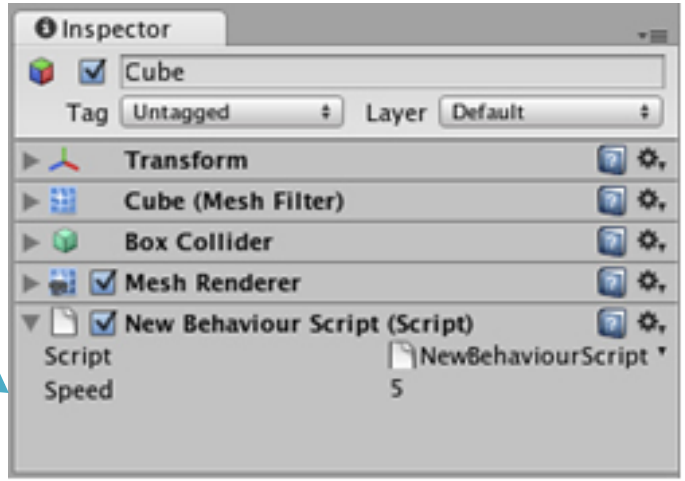
Les variables/attributs peuvent être

- `private` (accessibles uniquement dans le bloc où elles sont définies : fonction ou script)
- `public` (par défaut, accessibles dans tout le script et depuis l'extérieur)

En particulier, les attributs publics

- Apparaîtront dans l'Inspector et pourront être initialisées ou modifiées par l'utilisateur à l'édition ou au runtime
- Pourront également être initialisées ou modifiées par d'autres scripts (cf section Communication entre scripts)

```
public float speed = 5.0f;
void Update () {
    transform.Rotate(0, speed, 0);
}
```



Remarques:

- *Il est possible de masquer une variable globale publique de l'Inspector à l'aide de la ligne `@HideInspector` lors de sa définition dans le script*
- *Il est préférable d'avoir des variables privées que l'on rend modifiables dans l'inspector avec le modificateur `[SerializeField]`*

Debug

Basique = affichage console

Debug.Log(), Debug.LogError(), ..., print()

Avancé

MonoDevelop -> Breakpoints -> Run -> Attach to process -> Unity

Unity -> Play

Éléments de syntaxe (C#)

Les types simples

`float, bool, int...`

Les tableaux ("built-in arrays") :

Déclaration : `GameObject[] t;`

Initialisation : `t = new GameObject[10];`

Accès : `t[i] = myObject;`

Parcours :

```
for ( i = 0 ;  i < t.Lenght ; i++ )
foreach ( GameObject o in t )
```

Les classes utiles : `Vector3, Random, Math…`

Attention :

Les classes existantes utiles aux objets => Majuscule : `GameObject, Transform, Physics, Input…`

Les composants courants d'un objet => minuscule : `gameObject, transform, renderer, rigidbody…`

Input

http://docs.unity3d.com/ScriptReference/Input.html

Gestion des commandes clavier/souris

Paramètres et configuration des touches : Edit -> Project settings -> Input

Méthodes pour récupérer les commandes depuis les scripts :

float GetAxis (string axisName)	Returns the value of the virtual axis identified by axisName.
float GetAxisRaw (string axisName)	Returns the value of the virtual axis identified by axisName with no smoothing filtering applied.
bool GetButton (string buttonName)	Returns true while the virtual button identified by buttonName is held down.
bool GetButtonDown (string buttonName)	Returns true during the frame the user pressed down the virtual button identified by buttonName.
bool GetButtonUp (string buttonName)	Returns true the first frame the user releases the virtual button identified by buttonName.
bool GetKey (string buttonName)	Returns true while the user holds down the key identified by name. Think auto fire.
bool GetKeyDown (string buttonName)	Returns true during the frame the user starts pressing down the key identified by name.
bool GetKeyUp (string buttonName)	Returns true during the frame the user releases the key identified by name.
bool GetMouseButton (int button)	Returns whether the given mouse button is held down.
bool GetMouseButtonDown(int button)	Returns true during the frame the user pressed the given mouse button.
bool GetMouseButtonUp(int button)	Returns true during the frame the user releases the given mouse button.

Fonctions Monobehaviour

Mouse Events

OnMouseEnter()	called when the mouse entered the GUIElement or Collider.
OnMouseOver()	called every frame while the mouse is over the GUIElement or Collider.
OnMouseExit()	called when the mouse is not any longer over the GUIElement or Collider.
OnMouseDown()	called when the user has pressed the mouse button while over the GUIElement or Collider.
OnMouseUp()	called when the user has released the mouse button.
OnMouseUpAsButton()	only called when the mouse is released over the same GUIElement or Collider as it was pressed.
OnMouseDrag()	called when the user has clicked on a GUIElement or Collider and is still holding down the mouse.

http://docs.unity3d.com/ScriptReference/MonoBehaviour.html

TP

Programmation 1ère animation

Programmation interactions clavier souris

Création dynamique d'objets

"dynamique" = "pendant l'exécution"

Le moyen le plus simple est de créer des Prefabs depuis l'éditeur, comportant toutes les propriétés voulues, puis, dans un script, de les instancier dynamiquement et de modifier éventuellement les propriétés des instances (couleur, position, parent…).

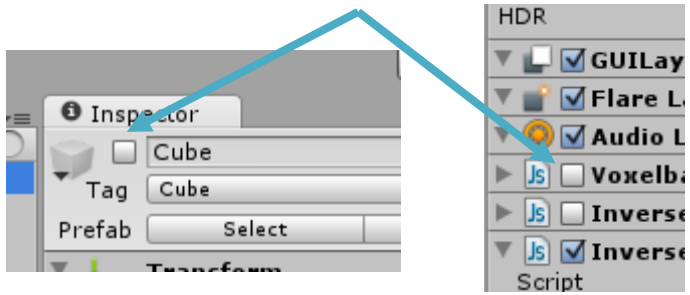
```
Transform monInstance = (Transform) Instantiate(monPrefab,
     position, transform.rotation);
monInstance.renderer.material.color = …
monInstance.parent = …
Destroy(monInstance.gameobject);
```

Rq : si `c` est un component et non un GameObject `destroy(c)` le supprime de l'objet

http://docs.unity3d.com/Manual/InstantiatingPrefabs.html

Activation dynamique d'objets/composants

Via l'inspector :



GameObject

`GameObject.SetActive(bool)`

Tests :

```
if (GameObject.activeSelf == …) …
x = GameObject.activeInHierarchy
```

Composant

Test ou affectation de booléen :

```
monComposant.enabled = true;
if (monComposant.enabled)...
```

Rq : `GameObject.active` et `GameObject.SetActiveRecursively()` ne sont plus utilisés (deprecated)

Communication entre scripts

Trouver le GameObject

Propriétaire du script : `gameobject`

```
GameObject.Find(string name)
GameObject.FindWithTag(string tag)
GameObject[] GameObject.FindGameObjectsWithTag(string tag)
```

Trouver le composant d'un GameObject o

Accès direct aux composants de base :

```
o.transform
o.renderer.material…
```

Accès direct aux composants par nom :

```
o.getComponent<MonScript>()
```

http://docs.unity3d.com/356/Documentation/ScriptReference/index.Accessing_Other_Components.html

Éléments de physique

Edit->Project Settings->Physics

Propriétés physiques : Collider & Rigidbody

Collider :

- Volume englobant ou mesh adapté à l'objet
- Permet la détection des collisions entre objets
- Prend en compte l'orientation des faces
- Peut combiner des colliders simples pour s'adapter à des objets complexes

Rigidbody :

- Permet le calcul des effets des collisions sur les objets
- Utilisation du moteur PhysX : gravité, rebonds…
- Propriétés : gravité, contraintes de position, de rotation…
- Nécessite un collider
- Pas de RigidBody = pas de physique

Propriétés physiques : scripts

Messages de collision (cf. Collision matrix et Events)

- "Message" envoyé automatiquement aux 2 objets en collision si au moins 1 des 2 possède un RigidBody
- Réception du message par scripts : OnCollisionEnter(), OnTriggerEnter()…
- Si un collider est défini comme isTrigger
  - Message envoyé (utilisé principalement pour déclencher des événements)
  - Pas d'effet du moteur physique sur l'objet

Rigidbody

- Peut être manipulé directement par scripts (addForce(), velocity…) mais éviter de le faire directement par son Transform (position…)

Cas particuliers

GameObject : Static

- Collisions sans physique (« Static Collider » = Collider mais pas de RigidBody)
- Exemples : environnement, murs…
- Ne doit pas être manipulé à chaque frame : coûteux + comportement indéfini avec les autres colliders

Rigidbody : isKinematic

- Pas d'effets des forces, collisions et gravité
- Manipulation par son Transform
- Exemples
  - Un objet parfois contrôlé par l'utilisateur, parfois par le moteur physique (explosion…)
  - Un objet qui peut pousser les autres sans être poussé

Fonctions Monobehaviour

Collisions

`void OnCollisionEnter (Collision collisionInfo)`	called when this collider/rigidbody has begun touching another rigidbody/collider.
`void OnCollisionExit (Collision collisionInfo)`	called when this collider/rigidbody has stopped touching another rigidbody/collider.
`void OnCollisionStay (Collision collisionInfo)`	called once per frame for every collider/rigidbody that is touching rigidbody/collider.

http://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Fonctions Monobehaviour

Collisions (triggers)

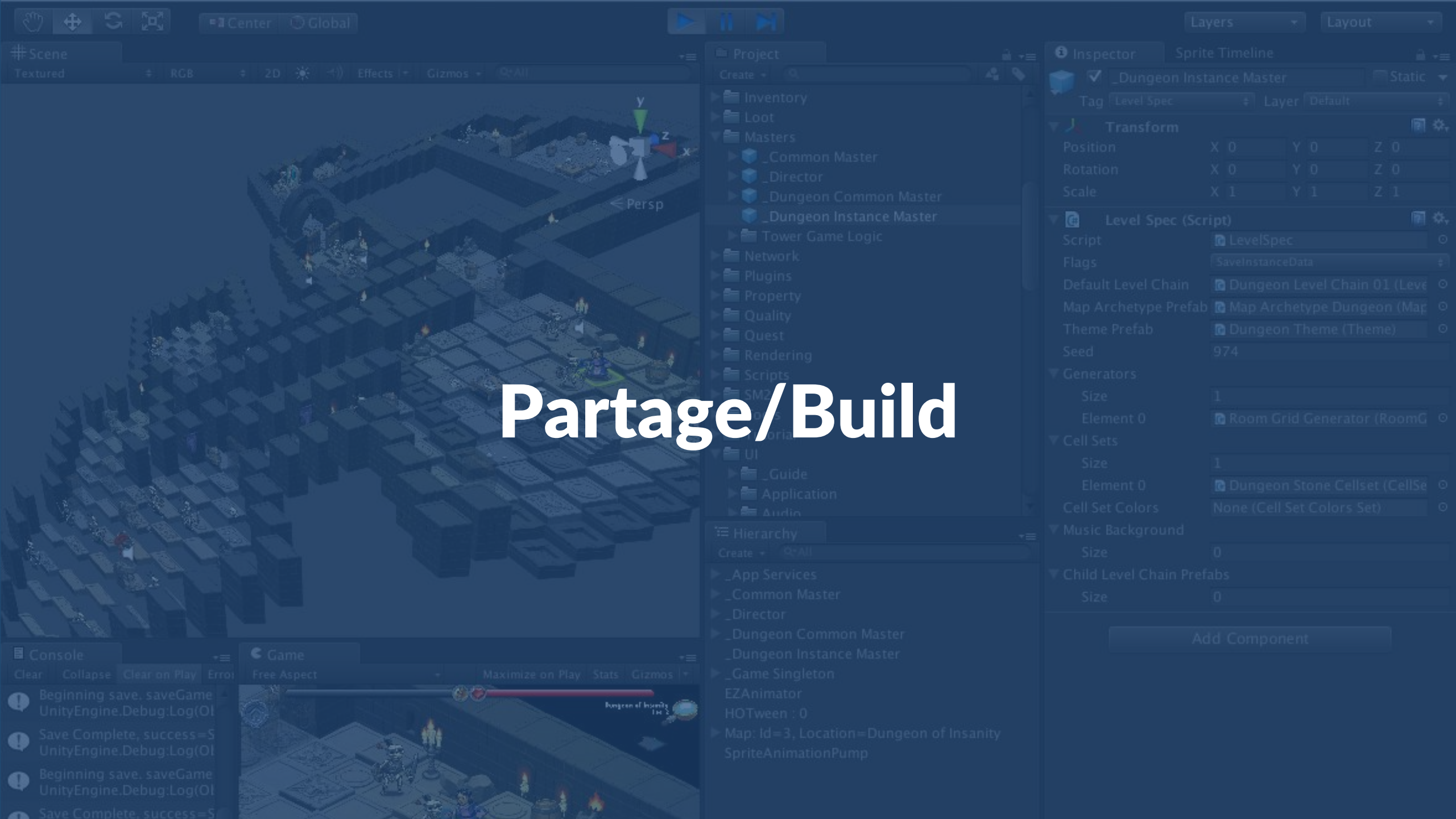
`void OnTriggerEnter (Collider other)`	called when the Collider other enters the trigger.
`void OnTriggerExit (Collider other)`	called when the Collider other has stopped touching the trigger.
`void OnTriggerStay (Collider other)`	called *almost* all the frames for every Collider other that is touching the trigger.

http://docs.unity3d.com/ScriptReference/MonoBehaviour.html

Collision matrix

Collision detection occurs and messages are sent upon collision						
	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider		Y				
Rigidbody Collider	Y	Y	Y			
Kinematic Rigidbody Collider		Y				
Static Trigger Collider						
Rigidbody Trigger Collider						
Kinematic Rigidbody Trigger Collider						

Trigger messages are sent upon collision						
	Static Collider	Rigidbody Collider	Kinematic Rigidbody Collider	Static Trigger Collider	Rigidbody Trigger Collider	Kinematic Rigidbody Trigger Collider
Static Collider					Y	Y
Rigidbody Collider				Y	Y	Y
Kinematic Rigidbody Collider				Y	Y	Y
Static Trigger Collider		Y	Y		Y	Y
Rigidbody Trigger Collider	Y	Y	Y	Y	Y	Y
Kinematic Rigidbody Trigger Collider	Y	Y	Y	Y	Y	Y


Partage/Build

Sauvegarde/Partage

Partage de certains éléments

Menu Assets -> Import/Export packages

On peut choisir les différents éléments du projet (scripts, prefabs…) à exporter dans un .unitypackage

Un certain nombre de packages de base sont importables (character controller, particules, végétation…)

Attention aux dépendances : vérifier que le package contient tout ce qui est nécessaire au projet par une réimportation (prefabs, scripts associés aux objets…) et/ou qu'un readme précise les autres paramètres (input manager, tags…)

Partage de tout le dossier Projet

Supprimer du projet tous les assets inutiles !

Effacer les dossier Library et obj qui peuvent être régénérés à l'ouverture

Génération de l'application

File -> Build settings

Sous Windows

- Exécutable *Projet.exe*
- Dossier de données *Projet_Data*

Web player

- Créer un dossier vide
- Page *Projet.html*
- Programme *Projet.unity3d*
- Nécessite plugin Unity Player dans le navigateur

TP

Build

Bilan

\+

Puissant (moteur de rendu, moteur physique…)
Accessible (composants, langage de script…)
Compatible HMD
Génération d'exécutable
Bien documenté et grande communauté
Version gratuite

\-

Pas de modèles 3d
Pas de CAVE ou périphériques de RV en natif

Du moteur de jeu au « moteur de RV »

Gestion de plusieurs aspects de l'application

- Rendu graphique
- Rendu audio
- Gestion de l'architecture matérielle
- Gestion des interactions
- Simulation de l'environnement
- API

En particulier

- Gestion du rendu stéréoscopique (caméra asymétrique)
- Gestion du multi-écrans
- Support des applications distribuées (clustering)
- Gestion des périphériques et du tracking (interfaçage VRPN)